371

N90-20671

DISTRIBUTED EARTH MODEL / ORBITER SIMULATION

Erik Geisler / IBM
Scott McClanahan / Ford Aerospace
Dr. Gary Smith / IBM

NASA Johnson Space Center
Workstation Prototype Lab
FS-7
Houston, Texas 77058

ABSTRACT

Distributed Earth Model / Orbiter Simulation (DEMOS) is a network based application developed for the UNIX environment that visually monitors or simulates the Earth and any number of orbiting vehicles. Its purpose is to provide Mission Control Center (MCC) flight controllers with a visually accurate three dimensional (3D) model of the Earth, Sun, Moon, and orbiters, driven by real time or simulated data. The project incorporates a graphical user interface, 3D modelling employing state-of-the art hardware, and simulation of orbital mechanics in a networked / distributed environment. The user interface is based on the X Window System and the X-Ray toolbox. The 3D modelling utilizes the Programmer's Hierarchical Interactive Graphics System (PHIGS) standard and Raster Technologies hardware for rendering / display performance. The simulation of orbiting vehicles uses two methods of vector propagation implemented with standard UNIX / C for portability. Each part is a distinct process that can run on separate nodes of a network, exploiting each node's unique hardware capabilities. The client / server communication architecture of the application can be reused for a variety of distributed applications.

1. INTRODUCTION

This paper describes a graphics project under development by the NASA / Johnson Space Center (JSC) Workstation Prototype Lab (WPL) staff that provides a scene generation tool capable of maintaining and displaying a realistic model of the Earth and various orbiting objects. Display output may be used to drive a large screen projector or closed circuit TV. The four major components of the application will be described. The first section covers the architecture and communication between the different tasks. The second section describes the user interface that controls the system. The third section is the model manager, which is the center of the application that manipulates the 3D graphics and coordinates the simulations. The final section discusses the simulation task, which generates positional and attitude data representing an orbiting vehicle.

2. BODY

2.1. Architecture

DEMOS is based on a server/client model. The model manager is the focal point of the system. It performs the server function, servicing requests from the client processes. The clients include one user interface task and several simulation tasks. There is one simulation task per orbiting vehicle, and several vehicles may be viewed simultaneously.

The processes that comprise DEMOS are Local Area Network (LAN) transparent, as a result, each task may run on different network nodes. Communication between the tasks is accomplished by passing packets via UNIX sockets, which is compatible across multiple vendor workstations. The sockets also work within a single workstation, so full flexibility is provided in defining the topology of the application. Configuration of each task's node can be defined by the user at run time.

The application may be distributed over multiple workstations to off-load computations to machines more appropriate for that type of work. The model manager must run on the workstation containing the target graphics hardware. Eliminating nearly all other processes on the model manager workstation, allows it to run at real time priorities, thus allowing the 3D image updates to occur more frequently. The graphical user interface is dependent on the X Server, graphics hardware, keyboard, and mouse, but it does not use much CPU, so a low end workstation is acceptable. The simulation tasks are CPU bound and profit from floating point hardware.

Figure 1 shows the system configuration of DEMOS. Circular components denote system processes. Double ended arrows represent communication between processes. Rectangluar boxes represent external data files. The "Config Data" contains system initialization information. The "Model Defs" contain external scene descriptions and model geometry.

These files are read by the model manager in order to construct a hierarchical scene. The user interface process is started first and employs the services of the X server. Upon successful initialization, the user interface process starts the model manager. The model manager in turn starts the Sun, the Moon, and any number of orbiting vehicle simulations. The system is shutdown in reverse order. The model manager terminates all simulation processes before terminating itself. The user interface is shutdown immediately upon a user request.

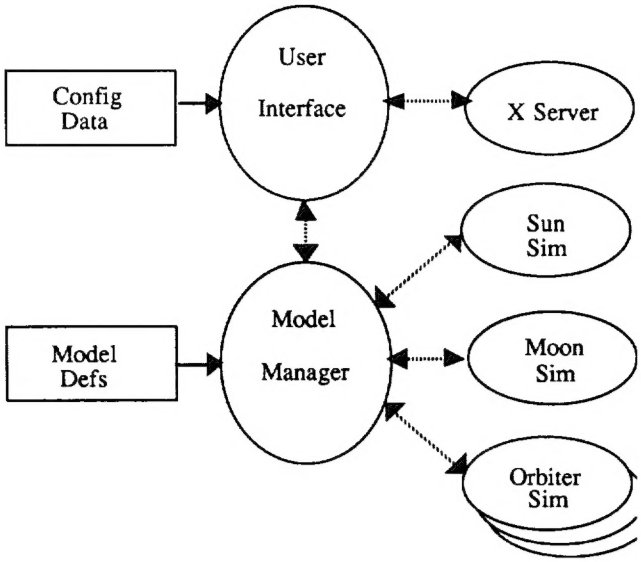

Figure 1 - DEMOS System Configuration

Each task in DEMOS uses the same packet send and receive subsystem. This prevents any task from blocking for I/O on socket operations. Since many packets may be sent at once, a queue holds the packets that the UNIX kernel cannot keep internally. The socket "streams" protocol is used to guarantee packet delivery and ordering. The queuing system also handles sending a partial packet. These packets can vary in size. At the receive end, the same subsystem isolates the application from incomplete packets by building and returning only complete packets.

2.2. User Interface

The user interface task provides the single point of contact between a user and DEMOS. It provides full control of DEMOS, including the ability to initiate and shutdown the system. The user interface does not have to be present after initializing DEMOS. The user can logoff the user interface and DEMOS will continue running. The system allows only one user interface to run at a time to ensure system consistency. System advisories are normally sent to the user interface, but if it is not present, then they are queued by the model manager until a user logs on.

There are two versions of the user interface for DEMOS - a graphical version, "xruif", and a command line interpreter version, "shuif". Xruif is based on the X Window System, so it is dependent on a graphics terminal controlled by an X server. On the other hand, shuif will run on any ASCII terminal, allowing more portability. The only difference between shuif and xruif is the user interaction. The low level areas of the two user interfaces share the same services.

2.2.1. Shuif

The command line interpreter version of the user interface is similar to the UNIX shell ("sh" or "csh"). It prompts for input from the keyboard, parses the input line for the command, and then executes the command.

A generic command line interpreter subsystem was created in the process of developing shuif. The first word of the input line is taken as the command name with the rest of the line being the arguments to the command. The command definitions, which are table driven, include the invocation, or callback, subroutine, as well as help information. The command line interpreter can be recursively nested to simulate submenus of commands. This subsystem also uses the "select" system call to block for I/O pending on any file descriptor. Each file descriptor has a corresponding callback routine which is called to process its data. I/O.

"Autotype" is a feature that allows shuif to run a complete user interface session in batch mode, reading commands from a file and echoing them to the screen as if they were typed by the user. This is useful for hands free demonstrations and test scripts. A "wait" command is included in shuif that suspends the user interface until an event occurs, such as waiting for the list of models to arrive before requesting a model to be loaded. Autotype files use the wait command to synchronize events within the system.

"Playback" is similar to autotype in that it allows the user working interactively with shuif to run a sequence of commands from a file. This is a convenience feature for redundant commands and to modularize operations involving a series of commands. Playback can also be invoked from an autotype file.

To complement the autotype and playback features, shuif can record (to a file) everything typed in by the user. Recording can be turned on or off at any time. Recording to an existing file appends the new commands.

Shuif provides commands for entering data for the simulation and base date values. When a simulation is started or the base date is set, the user has the option of using a data file or typing in all of the values. The data files can be created by a separate command that prompts the user for the values. All data files are validated when they are created and read.

2.2.2. Xruif

Xruif, the graphical user interface to DEMOS, is based on X11 Release 2. Since X is becoming a windowing standard, X clients are source code portable across many vendors' workstations. Xruif uses the X-Ray toolbox developed by Hewlett-Packard. The latest version of X-Ray is from the Release 3 tape of X11 from MIT. Xruif currently runs on a Sun 3/60 with the MIT X server in either monochrome or color.

The user interface style of xruif was not intentionally based on any existing application or style. It is based on the available X-Ray editors.

Xruif is composed of a single window divided into tiled panels. At the top is the title, followed by the main menu, then a work area, and an advisory panel at the bottom.

The work area is a reserved space where transient panels reside. All of the work area panels are the same size, even though their contents may not fill the panel. The work area panels are composed of a selection lists, (such as models, viewport configurations, viewport mappings and eyes, active vehicles, and the on-line help screens), or data entry screens for the base date and simulation values. These panels are activated by a main menu selection, and only one panel can occupy the work area at a time. When no panels are visible in the work area, a simple panel with the "work area" label in the center is left visible.

The user interface gadgets in X-Ray are called editors. Many of the X-Ray editors are used in xruif. The title bar editor contains a graphically offset single line of text with a selection box at each end. Each panel in xruif contains a title bar with a selection box containing a question mark for displaying help information on that panel. The push button editor is a matrix of oval buttons containing a label that is selected with the mouse. The main menu is comprised of push buttons. The list editor is a rectangle with an optional title bar at the top, optional scroll bars on the side, and a list of text that can be scrolled and selected (highlighted) with the mouse. This editor is used extensively in xruif. The text editor is a data entry field with a prompt to one side. It is used to enter simulation and base date values. A message box editor pops up a window containing an icon, some text, and some push buttons. It is used to force answering "Are you sure?" questions. A group box editor is simply a rectangle with a label at the top to surround a group of editors and visually associate them.

The title panel also uses the static raster editor to display pixmaps, such as the NASA logo, the WPL logo, and the DEMOS icon. The DEMOS icon is also used to represent the xruif window when it is closed.

There are several advantages to using separate panels. First, it modularizes each component of the user interface. Any panel can easily be modified and rearranged without affecting the other panels. Second, the X window events "entry" and "leave" are used to determine when the pointer goes into or out of a panel. This allows each panel to do its own input processing instead of having to handle all inputs in one routine. Third, each panel can size and create its own editors. Finally, panels can be redisplayed independently, each handling its own "expose" events.

Xruif employs an on-line help facility for information on each panel. The title bar of each panel has a help icon, which, when selected, brings up the help panel in the work area, overlaying the previous panel. When the help panel is terminated, the previous panel is restored. The help panel contains scrollable help text, plus a help index listing all help screens. Selection of a help index item displays that panel's help screen. The help screens are loaded at initialization from ASCII text files. For DEMOS, help screens were formatted by the "nroff" utility and can be easily customized by the user.

2.3. Model Manager

The model manager is responsible for servicing user interface requests, loading data models, managing simulations, and generating accurate visual displays of a modeled scene. Its implementation employs the PHIGS standard, as well as PHIGS+ extensions for lighting and shading. The Raster Technologies PHIGS+ subsystem off-loads a number of graphic functions including model hierarchy management, traversal, and rendering/display. Many functions are performed in firmware. Using this graphics architecture, the model manager is able to concentrate on a variety of control functions associated with managing multiple, asynchronous simulations. The model manager is composed of three major elements: Scene Construction, Simulation Management, and View Generation.

2.3.1. Scene Construction

After communications have been established with the user interface task, the model manager begins by constructing an in-memory tree representation of a selected scene hierarchy. A user selects a scene by choosing a top-level description file. The model manager reads this verb-based description file in order to build an internal representation of the scene. Description files may reference other description files. In this manner, a complex external model hierarchy may be defined. The model manager will recursively read these files until the entire scene tree is built. Using this technique, generic scenes may be developed and processed by a general modelling subsystem.

Besides model hierarchy construction, description files provide additional information which is attached to the model's geometric definition. This separation of model geometry and model attributes allows models to be tailored for rendering performance versus realism. Each file has a specified type, which determines how the remaining commands are to be interpreted. A variety of types are currently supported: 'model', 'eye', 'camera', 'scene', 'light', and 'ghost'.

A 'model' description file provides the following information: the model units, initial placement, display options (polygon, vector, polyline), shading method (flat, Gouraud), surface properties, color model, hidden-line/hidden-surface options and an optional reference to a data file containing the actual geometric model. Model attributes are inherited from parent models. Typically, a top level model node provides overall model information and attributes while children nodes reference individual submodels and define how they are geometrically related to their parent.

An 'eye' or 'camera' description file provides the definition of viewing parameters for a single viewpoint. The only distinction made between eyes and cameras is that cameras represent physical optical devices while eyes define a synthetic viewpoint. Both are treated as submodels, positioned relative to their parent node. By attaching eyes and cameras to a geometric model, a wide variety of views can be supported. This viewing mechanism forms a major element within the model manager. A majority of the model manager's computational effort is spent maintaining selected views. The following information can be defined for an eye or camera: camera position, camera orientation, perspective reference point, view distances (front, view, back), projection type (parallel, perspective), and viewing window parameters.

A 'scene' description file defines global scene characteristics. Specifically, it provides the following information: scene lighting method (ambient, diffuse, specular, none), true or pseudo color display indicator, background screen color, viewport edge characteristics, viewport titles flag, initial viewport definition(s) and background colors, screen aspect ratio, Normalized Projection Coordinate the (NPC) window and Device Coordinate (DC) viewport in which NPC window will be mapped. Many of the developed scenes refer to a common scene node since this information rarely changes. Changing the scene lighting and the number of viewports can drastically affect scene display rates. The DC viewport provides the capability to place the graphic display into a selected portion of the screen. This becomes important when the RGB signal is converted to video via converter boxes such as Genlock or RGB Technologies VideoLink.

A 'light' description file defines a single light source. Ambient, infinite, point, and spot light types are supported. Depending on the light type, a number of lighting characteristics may be defined, such as color, location, direction, concentration exponent, and cone of influence. Adding additional lights seems to have only a minimal computational effect on the overall rendering process. DEMOS currently employs a single infinite light source - the Sun. Additional lights might be added to have the orbiter always visible to the user even though it is positioned on the dark side of the planet.

A 'ghost' description file defines an object which assumes and maintains a position relative to its immediate parent node. As its name implies, a ghost object is an invisible object which cloaks another object. Ghost objects are semi-attached to their parent. This is, they only receive positional updates; attitude transformations are not applied. These type of objects are typically used to establish a set of viewpoints associated with an orbiting vehcile. Since these viewpoints only accept positional updates, and therefore move along with an object, they are capable of viewing rotational (attitude) changes to the object in which they are connected. This feature provides a flexible viewing mechanism and is used to support visual verification of spacecraft orientation, as well as, Earth rotations from a point in space.

During description file processing and hierarchical scene tree construction, a set of linear lookup tables are developed in order to minimize the model editing process. Each table entry contains a unique object name followed by a tree node pointer. The use of this pointer eliminates unnecessary tree traversals by the model manager when updating an object's position and attitude. In addition, the PHIGS structure ID is obtained from the tree node, and is used to perform PHIGS editing. The reason in-memory scene tree structures are edited along with the PHIGS structures is to facilitate the formation of a particular view from an eye or camera. The PHIGS specification allows structure inquiries to obtain this information; however, the PHIGS implementation currently used does not support this operation. Combining an in-memory tree structure and the sorted lookup tables provides an efficient framework for model editing. The linear table provides efficient model searching while the in-memory tree structure provides the necessary model hierarchy.

After the in-memory scene tree has been constructed, it is loaded into the PHIGS Central Structure Store (CSS). The CSS provides a central database where graphics information is stored and edited. In order to construct the PHIGS database in a contiguous manner, the model manager recursively traverses the in-memory scene tree and loads each model and light node into the CSS. If a child node representing a model, ghost, or light is referenced within the current node, a PHIGS structure execution command is issued to link

this child node to its parent. Eye and camera nodes are ignored during this PHIGS loading process and are managed separately.

2.3.2. Simulation Management

The processing architecture of the model manager is based on a state machine approach employing time and events. To manage multiple, asynchronous simulations, the model manager must maintain its own internal clock. This clock is established by the user issuing the system start time command. Once the time is set, the model manager begins propagating it by a discrete unit. In addition, the model manager automatically spawns a Sun and Moon simulation task on previously defined workstations. The user also determines how quickly time should propagate and the amount of Earth rotation per display update. This clock is used to synchronize all events within the model manager.

Simulations are initiated upon reciept of a user interface request. The model manager retains the specified simulation information within an internal state structure. These structures hold and maintain information neccessary for communications, groundtrack requests, and position and attitude requests. States transition from one state to another due to an occurrence of an event. For example, a simulation is not started until the internal clock is equal to or greater than the simulation starting time. Once started, the simulation, or monitoring element, transitions from the 'wait_to_start' state to the 'has_started' state. Typically, simulations enter a cyclic state where the model manager continually requests their next position for the current time of interest (e.g., the internal system time). Since the model manager makes all the requests, it controls the rate at which simulation elements respond. Simulation or monitoring elements never send unsolicited information. This greatly simplifies their control. In effect, simulation management is handled via a master / slave approach rather than with the client / server relationship held with the user interface. This control technique also ensures the model manager is never inundated with data from clients' simulations.

The notion of a time node was developed to maintain an accurate visual display of multiple moving objects. When time is propagated, a node is allocated and placed on the end of a time list. For each time unit, a request is generated for each active simulation element in order to update its position, attitude, or light direction. These requests are attached to the current time node. When the simulation element responds with appropriate data, the corresponding model is updated to reflect this update, and the request is removed from the appropriate time node. Once all requests for a particular time node have been removed, the time node is freed and the scene is in a correct state for the next display. If all requests for a time node have been removed, and an earlier time node still contains outstanding requests, then these earlier time nodes are destroyed - leaving only the latest information. By employing time nodes and multiple requests per time unit, the accuracy of the visual display is ensured.

2.3.3. View Generation

The model manager spends the majority of its processing maintaining accurate visual representations of the scene being modeled. It supports a wide array of scene viewing capabilities. Under user control, the graphics screen may be partitioned into a number of viewports. Each viewport is treated as an empty slot in which an eye or camera may be assigned. Only one view (eye or camera) may be assigned to a particular viewport at any one time; however, a view may be assigned to multiple viewports. Viewports have a background color, and are outlined to indicate their screen coverage. In addition, viewports have the property of visibility. The user may wish to temporarily turn off a particular viewport to improve display rates or to ignore uninteresting views. Viewports which have an assigned eye or camera may have a small title displayed to help identify the particular view. These titles are extracted from the corresponding eye or camera nodes from within the scene tree.

During the scene tree construction phase, a default viewport configuration file and a default view assignment file are read to provide an initial viewing framework. This framework is used to view the scene prior to starting simulations. Once a scene is loaded, the user interface requests a list of all available viewport configurations and their current view assignments. Given this information, the user may freely assign views to viewports, toggle viewport visibility, or select another viewport configuration.

The model manager constructs a view for a given viewport in the following manner. First, a view is assigned to a particular viewport by copying the specified viewing parameters to the desired viewport data structure. A view mapping matrix is then computed for this viewport. The next step involves the actual view generation, given an arbitrary viewing position in modelling coordinates. Since the eye coordinate system is fixed, it is necessary to transform the world coordinate system into this eye coordinate system. The scene tree contains all information concerning model hierarchy, and is therefore used to compute this transformation by traversing the scene tree backward from the eye or camera node to the tree's root node. Initially, the eye's orientation is set to the identity matrix. This matrix is then transformed by applying inverse transformations while traversing up the tree. Once the root node is reached, a final orientation matrix has been formed, and it is then associated with the corresponding viewport.

The viewing computation is then completed by loading the newly computed viewing represen-

tation and allowing PHIGS to traverse the hierarchical model contained in the CSS. In order to minimize the view construction, only the assigned, visible, viewports are computed.

2.4. Simulation Components

The simulation tasks provide the model manager with the necessary data to maintain an accurate representation of the Sun, the Moon, any number of orbiting vehicles, and the orientation of the Earth within the M50 coordinate system (the basic JSC inertial coordinate system).

Currently, three types of simulation tasks are supported: a Sun simulation, a Moon simulation, and an orbiting vehicle simulation. Simulation tasks are started by the model manager via a remote procedure call. They are typically deployed on workstations providing floating point hardware. Once a simulation has successfully started and has established communication with the model manager, it is sent a packet containing all information required to begin processing.

A simulation component of DEMOS consists of up to five functional elements:

1) Compute the Rotation-Nutation-Precession (RNP) matrix. The RNP matrix relates the M50 coordinate system to a coordinate system fixed to the Earth.
2) Generate from one to ten orbits worth of ground tracks for the orbiting vehicle
3) Determine the position over the Earth of the orbiting vehicle.
4) Determine the attitude of the orbiting vehicle axes relative to an Earth-fixed coordinate system so that the vehicle maintains a pitch, roll, and yaw of zero degrees relative to the UVW local orbital reference frame (U is a unit vector in the direction of the radius vector, W is a unit vector in the direction of the angular momentum vector, and V is the unit vector which forms a right-handed system). (Note that the body axis system for this application has the x-axis out the nose of the orbiter, the z-axis out the top of the orbiter, and the y-axis out the left wing).
5) Determine the location of the Sun and the Moon.

2.4.1. Computation of the RNP Matrix [12]

The fundamental transformation matrix for the simulation component is the RNP matrix. It incorporates all of the precession, nutation and rotation changes that have affected the orientation of the Earth in inertial space since 1950. It relates the orientation of an axis system fixed to the Earth relative to the M50 coordinate system. The user interface task provides the base-time-of-interest values. These include: the year, month, day, hour, minute and second. The time difference between Ephemeris Time and Universal Time Corrected is also provided.

Once the input base time has been obtained, the computation of the RNP matrix proceeds as follows:

1) Calculate the Julian Universal Date and the Julian Ephemeris Date.
2) Compute the three precession angles.
3) Compute the precession transformation matrix, P.
4) Compute the nutation angles.
5) Compute the nutation in longitude.
6) Compute the nutation in obliquity.
7) Compute the nutation transformation matrix, N.
8) Compute the rotation transformation matrix, R, which orients the X-axis (through Greenwich) for the base time of interest.
9) Compute the RNP matrix by multiplying the R, N, and P matrices together.
10) Perform the z-axis rotation to rotate the RNP matrix back to December 31, 0 hours, 0 minutes, 0 seconds of the previous year.

This fundamental RNP matrix is employed to transform an M50 vector (for a given time) into an Earth-fixed vector. This is extremely important for the generation of ground tracks or for positioning a vehicle over the surface of the Earth.

2.4.2. Generation of Ground Tracks

The user interface task provides the number of orbits worth of ground tracks that are to be displayed. This number is passed to the ground track simulation element where 180 sets of Earth-fixed latitude and longitude points are generated for each orbit. These points are passed to the model manager which then displays the ground tracks on the 3D Earth Model.

Each Earth-fixed latitude and longitude point is computed as follows:

1) Propagate the state vector to the particular time along the ground track (the delta time between propagation steps is the period of the orbit divided by 180 points).
2) Calculate an updated RNP matrix using the time of the state vector.
3) Transform the propagated M50 position vector into and Earth-fixed position vector using the RNP matrix.
4) Calculate the Earth-fixed latitude and longitude from the Earth-fixed position vector.

The ground track simulation element either uses a two-body propagation method or a modified Analytic Ephemeris Generator (AEG) propagation method to generate the state vectors from which the latitude and longitude points are computed. The user specifies which propagation method is desired when the initial state vector is entered.

2.4.3. Computation of Position

The user interface provides values for the initial simulation. The object could be the shuttle orbiter, the Space Station, or any other satellite of the Earth. The user decides the choice of units (e.g. feet, meters, or Earth radii) and the initial time of the 'state'. This 'state' can be entered either as M50 position and velocity vectors or as M50 Keplerian orbital elements (semi-major axis, eccentricity, inclination, longitude of the ascending node, argument of perigee and mean anomaly). Finally, the choice of propagation method is entered. This can be either two-body or AEG propagation. When the simulation element is initialized, additional orbital parameters are computed which will be utilized in the propagation of the position and velocity. In the case of two-body propagation [6], the additional parameters include unit vectors in the two-body orbital plane. An AEG propagation is initialized by computing a set of 'invariant' elements which can be used to propagate position and velocity including the effects of the J2, J3, and J4 gravitational harmonics of the Earth. The AEG propagation method is a scaled down version of Edgar Lineberry's Analytic Ephemeris Generator [8] which is used in the MCC to support missions. The scaled down version computes the effects of the short-period terms on the orbital elements. The version implemented for DEMOS does not include the calculation of drag effects.

When a time is given to the position computation element, Get_position, the position and velocity are propagated to that time. This time may be sent to Get_position either by the ground track computation element or by the model manager. Once the position and velocity are propagated, a series of coordinate transformation routines rotate these vectors into an Earth-fixed coordinate system. The Earth-fixed position vector is then used to calculate the Earth-fixed latitude and longitude.

2.4.4. Computation of Attitude [9]

After the position is obtained, the pitch, roll, and yaw attitude angles of the orbiting vehicle relative to the Earth-fixed reference frame are computed to maintain the vehicle attitude of its body axes relative to the UVW local orbital reference frame. This "UVW hold" attitude causes the shuttle to appear on the graphics screen with its nose parallel to the ground tracks and the plane of the wings perpendicular to the radius vector. This allows the payload bay doors to be visible to a viewer looking down on the shuttle as it orbits the Earth.

2.4.5. Computation of the Sun Position [2,3]

The model manager has the capability to provide lighting over the surface of the Earth, by knowing where the Sun is relative to the Earth-fixed coordinate system. The Sun position computation element, Get_sun, calculates the Earth-fixed position of the Sun by the following steps:

1) Calculate the precession angles and the precession matrix P, for the time of interest.
2) Calculate the Mean Longitude of Perigee for the Sun relative to the Mean Equinox of Date.
3) Calculate the Mean Anomaly for the Sun relative to the Mean Equinox of Date.
4) Calculate the eccentricity of the Earth's orbit around the Sun.
5) Solve Kepler's equation for the Eccentric Anomaly of the Sun.
6) Calculate the True Anomaly of the Sun.
7) Calculate the Mean Obliquity of the Ecliptic.
8) Calculate the longitude of the Sun in the Ecliptic plane.
9) Calculate the magnitude of the radius vector from the Earth to the Sun.
10) Compute the position vector of the Sun in Ecliptic coordinates.
11) Apply the precession matrix, P, to this Ecliptic vector to compute the position vector for the Sun in M50 coordinates.
12) Rotate this M50 position vector using the RNP matrix to Earth-fixed coordinates and extract the Earth-fixed latitude and longitude of the Sun.

2.4.6. Computation of the Moon Position [2,3]

The model manager can move one of its viewpoints sufficiently far from the Earth-Moon system so that both the Earth and the Moon are visible in the same view. If the system time is accelerated the Moon can be seen to orbit the Earth.

The Moon position computation element, Get_moon, calculates the Earth-fixed position of the Moon by the following steps:

1) Compute the precession angles and the precession matrix, P.
2) Calculate the nutation angles.
3) Calculate the Ecliptic latitude, Ecliptic longitude, and parallax of the Moon using Fourier Series Expansions (sine and cosine terms) of combinations of the nutation angles.
4) Compute the magnitude of the radius vector from the Earth to the Moon.
5) Compute the position vector of the Moon in the Ecliptic plane.
6) Rotate the Ecliptic position vector into the M50 coordinate frame using the precession matrix, P.
7) Rotate this M50 position vector using the RNP matrix to Earth-fixed coordinates and then extract the Earth-fixed latitude and longitude of the Moon.

Eventually, the position, velocity and attitude information for the orbiting vehicle will be obtained over the LAN from the Mission Operations Computer (MOC) or Calibrated Ancillary System (CAS). The internal units for the simulation component have been kept compatible with the Ground Based Space Sys-

tems (GBSS) internal units on the MOC to ease this transition.

3. CONCLUSION

DEMOS is a successful implementation of 3D modelling employing accurate simulations of the Earth, Sun, Moon, and any number of orbiting objects. It provides a visualization tool which has the capability to simulate / monitor orbiting objects and to display a realistic scene in an acceptable time period. A flexible viewing system allows flight controllers to view objects from a variety of viewpoints. Vehicle cameras and synthetic eyes may be defined to inspect spacecraft activity from arbitrary view positions. The distributed architecture provides the framework for future application extensions. Application software employs the latest workstation standards, maximizing its lifecycle while minimizing any rehosting costs. Simulation techniques are implemented from proven algorithms.

4. ACKNOWLEDGEMENTS

The authors wish to thank Randall Barnett of Lincom Corporation for his PHIGS assistance and software techniques. His optimized algorithms improved view generation times considerably. We would like to also thank the Mission Planning and Analysis Division (MPAD) graphics lab for their generous supply of high fidelity graphics models of various spacecraft, which helped assimilate realistic scenes.

5. REFERENCES

1. Computer Science Corporation, FLIGHT DYNAMICS / SPACE TRANSPORTATION SYSTEM 3-D MONITOR SYSTEM RELEASE 2 SYSTEM DESCRIPTION, CSC/SD-88/6066, Contract NAS 5-31500, Task Assignment 58 214, NASA Goddard SFC, Greenbelt, Maryland, August 1988.

2. Escobal, P. R., METHODS OF ASTRODYNAMICS, John Wiley & Sons, Inc., New York, 1968.

3. EXPLANATORY SUPPLEMENT TO THE ASTRONOMICAL EPHEMERIS AND THE AMERICAN EPHEMERIS AND NAUTICAL ALMANAC, H. M. Stationery Office, London, 1961.

4. Foley, J. D., Van Dam, A., FUNDAMENTALS OF INTERACTIVE COMPUTER GRAPHICS, Addison-Wesley Publishing Company, Philippines, 1982.

5. Gettys, Jim, Newman, Ron, Scheifler, Robert W., XLIB - C LANGUAGE X INTERFACE, Massachusetts Institute of Technology, Cambridge, MA, 1987.

6. Herrick, Samuel, ASTRODYNAMICS, Volume I, Van Nostrand Reinhold Company, London, 1971.

7. Hewlett-Packard Company, PROGRAMMING WITH THE XRLIB USER INTERFACE TOOLBOX, February 1988.

8. Lineberry, Edgar, INVARIANT ORBITAL ELEMENTS FOR USE IN THE DESCRIPTION OF MOTION ABOUT AN OBLATE EARTH, JSC Internal Note No. 74-FM-84, December 4, 1974.

9. MATHEMATICAL STANDARDS AND GUIDELINES, IBM GBS Programmer's Guide, Section 6, September 1, 1976.

10. Mortenson, M. E., COMPUTER GRAPHICS: AN INTRODUCTION TO THE MATHEMATICS AND GEOMETRY, Industrial Press Inc, New York, NY, 1989.

11. O'Reilly and Associates, XLIB PROGRAMMING MANUAL FOR VERSION 11 RELEASE 2 OF THE X WINDOW SYSTEM, Volume I, O'Reilly and Associates, Newton, MA, April 1988.

12. Schulenberg, C. W., DESCRIPTION OF A SELF CONTAINED SUBROUTINE WHICH ANALYTICALLY GENERATES INTERPLANETARY COORDINATE SYSTEM TRANSFORMATIONS REFERENCED TO THE MEAN OF 1950.0 EPOCH, TRW Note 70-FMT-853, Sept. 30, 1970.